Database Documentation

Date: 05-June-2011

From: Bernd Plumhoff, www.sulprobil.com

Subject: **System and User Manual for Database**

Table of Contents

1. SUMMARY 2

2. SYSTEM DOCUMENTATION 2

3. USER DOCUMENTATION 3

3.1 Super User with Read/Write Access 3

3.2 Normal User with Read-Only Access 4

4. APPENDIX 5

4.1 Database Definition 5

4.2 Write Interface to Database (via stored procedure & Excel) 9

4.3 Read Interface from Database (via stored procedure & Excel) 11

1. Summary

This database was implemented to

- Support selected processes with a simple “central” data repository
- Get rid of individual spreadsheets which have to be maintained with high effort
- Simplify the task of data retrieval and of data storage
- Enable additional calculations over defined periods of time and “slice & dice” calculations
- Precisely define user or process access rights

This database is a simple all-purpose database. After the initial setup there is no further need for database table maintenance or index maintenance (apart from deleting old records to avoid the database becoming too big).

Limitations:

- The database is not designed for mass storage – you are advised to define a regular data deletion (maintenance) procedure for each new field / data you regularly add to this database
- The design is simple and field orientated. There is no support for curves or for surfaces – which means you cannot store nor retrieve curves or surfaces in one go. All points would have to be dealt with individually. This is possible but it creates quite an overhead to the database engine. So if this needs to be used it should only be temporary and not for mass storage
- Fields are stored as string values. The user needs to convert all data into the required data types

There are 3 levels of access or responsibilities:

- Technology and help desk support – After initial setup the database is backed up on a daily basis, repair or restore can be done within two hours, new users with read or read/write access can be introduced or old users can be deleted within 24h
- Super user with read/write access – YOU define new fields and feeds the database with approved rate sources
- Other Users with read-only access

YOU are the owner of this database.

2. System Documentation

Technology is responsible for the database setup, user setup and for repair or restore of the database. And Technology has to ensure that this database is up and running with a monthly average availability of 99% during normal working hours. New users with read-only or read/write access can be introduced or old users can be deleted within 24h.

Since YOU are the owner of this database, all access right changes have to be approved by YOU. Currently only YOU should have read/write access to the database.

For setup details refer to the Appendix, Database Definition. Please notice that the database consists of only two tables and two stored procedures. The two stored procedures provide a sophisticated write and read access to the database. Since they are stored on the server side the client (user) does not need to invest a big effort but he can easily make use of them. The database structure is not intended to change over time. This means that the database maintenance effort is reduced to a possible minimum: Only if the database gets too big old records will have to be deleted. If the write access user (YOU) defines a regular maintenance procedure which deletes old data records for all fields which are regularly added to the database the effort on top of this should be zero.

Please notice: Technology should be able to rebuild or to recreate this database within two hours. This includes corruption of database indices.

3. User Documentation

The access structure to the database is fairly simple: There is a class of super users with write access. This is restricted to YOU only.

3.1 Super User with Read/Write Access

YOU are the owner and the only user with write access to the database. This can be reflected by a relevant AD group. This is not intended to be enhanced for other super users. YOU are responsible for all field definitions and for all feeds into the database. Later we might need to delete old unnecessary records, too. YOU are the only party to approve additional users or super users.

Please notice the limitations of this database:

- The database is not designed for mass storage
- The design is simple, field orientated. There is no support for curves or for surfaces – which means you cannot store nor retrieve curves or surfaces in one go. All points would have to be dealt with individually. This is possible but it creates quite an overhead to the database engine. So if this needs to be used it should only be temporary and not for mass storage
- Fields are stored as string values. The user needs to convert all data into the required data types

For the Excel write interface into the database refer to Appendix, Write Interface to Database.

These applications are feeding the database (Example):

No	Application	Source	Field	Frequency	Comment
1	Load_DB	Xtrakter	PRICE_MID	Daily	
2	Load_DB	Xtrakter	QUOTE_ASK	Daily	
3	Load_DB	Xtrakter	QUOTE_BID	Daily	
4	Load_DB	Xtrakter	NbMktMkr	Daily	Number of Market makers
5	Load_DB	Xtrakter	TrdBandLow	Monthly	Minimal number of trades
6	Load_DB	Xtrakter	VolData	Monthly	Trade volume (local ccy)
7	Load_DB	Markit	CompositePrice	Daily	**All Markit data older than 60 days gets deleted**
8	Load_DB	Markit	Depth	Daily	Number of price providers, **All Markit data older than 60 days gets deleted**

3.2 Normal User with Read-Only Access

For the Excel read interface from the database refer to Appendix, Read Interface from Database. This can be reflected a the relevant AD group.

Examples to retrieve static information:

The Excel function call
=sb_get_param("BPIZC610","PX_CLOSE_1D","20110121","RILO")
would result in
29.205
Please note that this value is a string value! In order to get a number of type double you would need to convert the result yourself, for example:
=--sb_get_param("BPIZC610","PX_CLOSE_1D","20110121","RILO")

The Excel function call
=sb_get_param("FR0010850719","SECURITY_NAME","19000101","Bloomberg")
would result in
COFP4.379 02/17

Static data is stored with the date 1-Jan-1900. Please notice that the database would also have returned this result if you had called it with a younger date than that. This is because the sb_get_param function returns the youngest entry which is older or equal to the request date. You would need to use the sb_get_paramarray function to retrieve the complete database record (date included) to check the exact date of the returned value.

Example to retrieve market (dynamic) data:

The Excel function call
=sb_get_param_array("US172967EZ03","PRICE_MID","20101223","Xtrakter")
would result in
{"US172967EZ03","PRICE_MID","Xtrakter",40535,"102.172"}
Here 40535 is the numerical date of 23-Dec-2010.

4. Appendix

4.1 Database Definition

```
CREATE TABLE [dbo].[param] (
	[identifier] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[param] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[source] [varchar] (30) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[fromDate] [datetime] NULL ,
	[toDate] [datetime] NULL ,
	[value] [varchar] (500) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[updTime] [datetime] NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[param_details] (
	[source] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[param] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[idYN] [char] (1) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[rename] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[priority] [int] NULL ,
	[inputYN] [char] (1) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO


 CREATE  UNIQUE  INDEX [param_pk] ON [dbo].[param]([identifier], [param], [toDate], [fromDate], [source]) ON
[PRIMARY]
GO

 CREATE  INDEX [param_val] ON [dbo].[param]([param], [value], [identifier], [toDate]) ON [PRIMARY]
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO

CREATE proc get_param
	@identifier		varchar(100),
	@param		varchar(100),
	@source		varchar(30),
	@dated		datetime
as
	if @identifier > ''
	begin
		select	identifier, param, source, fromDate, value
		from	param
		where	identifier = @identifier
			and param like isnull( @param,param )
			and source like isnull( @source,source )
			and (	@dated >= fromDate
				and @dated < isnull( toDate, '1 jan 3000' )
				or @dated is null and toDate is null )
		union
		select	identifier, param, source, fromDate, value
		from	param
		where	identifier = @identifier
			and param like isnull( @param,param )
			and source like isnull( @source,source )
			and fromDate is null
	end
```

```
	else
	begin
		select	identifier, param, source, fromDate, value
		from	param
		where	param like isnull( @param,param )
			and source like isnull( @source,source )
			and (	@dated >= fromDate
				and @dated < isnull( toDate, '1 jan 3000' )
				or @dated is null and toDate is null )
		union
		select	identifier, param, source, fromDate, value
		from	param
		where	param like isnull( @param,param )
			and source like isnull( @source,source )
			and fromDate is null
	end
GO
CREATE procedure [dbo].[set_param]
	@identifier			varchar(100),
	@param			varchar(100),
	@source			varchar(30),
	@dated				datetime,
	@value				varchar(500),
	@stopLoop			char(1) = 'n'
as
	set nocount on

	declare	@vf		datetime,
		@vt		datetime,
		@rename			varchar(100),
		@priority	int


	-- ignore if bad params
	if	isnull( @param,'' ) = ''
		or isnull( @identifier,'' ) = ''
		or isnull( @value,'' ) = ''
		or isnull( @source,'' ) = ''
		or @value like '#N/A%'
		or @value ='?'
	begin
		return
	end

	select	@rename = rename,
		@priority = priority
	from	param_details
	where	source = @source
		and param = @param
		and @stopLoop = 'n'

	if @rename > ''
	begin
		if not exists(
			-- an existing value from a higher priority source
			select	1
			from	param p1
			where	p1.identifier = @identifier
				and p1.toDate is null
			   and p1.fromDate >= @dated
				and p1.param in (
					select	param
					from	param_details
```

```
                                        where   source=@source
                                                and rename = @rename
                                                and param != @param
                                                and priority < @priority ) )
        begin
                exec set_param @identifier, @rename, @source, @dated, @value, 'y'
        end
end


if @source = 'input'
begin
        select  @dated = getdate()
end


-- if static value (ie not time dependant)
if @dated is null
begin

        -- if value unchanged then return
        if exists(
                select  1
                from    param
                where   identifier = @identifier
                        and param = @param
                        and source = @source
                        and value = @value
                        and fromDate is null
                        and toDate is null )
        begin
                return
        end

        -- update value to new value if it exists
        update  param
        set     value = @value,
                updTime = getdate()
        where   identifier = @identifier
                and param = @param
                and source = @source
                and fromDate is null

        -- insert new value if not
        if @@rowcount = 0
        begin
                insert  param (  identifier, param, source,   fromDate, toDate, value,   updTime )
                values  ( @identifier, @param, @source, null, null, @value, getdate() )
        end
end
else
begin
        -- if value unchanged then return
        if exists(
                select  1
                from    param
                where   identifier = @identifier
                        and param = @param
                        and source = @source
                        and @dated >= fromDate
                        and @dated < isnull( toDate, '1 jan 3000' )
                        and value = @value )
        begin
```

```
            return
        end

        -- get dates of existing record
        select  @vf = fromDate,
                @vt = toDate
        from    param
        where   identifier = @identifier
                and param = @param
                and source = @source
                and @dated >= fromDate
                and @dated < isnull( toDate, '1 jan 3000' )

        -- if before any current records
        if @vt is null
        begin
                -- get next date
                select  @vt = min( fromDate )
                from    param
                where   identifier = @identifier
                        and param = @param
                        and source = @source
                        and fromDate > @dated
        end

        -- update fromDate of any existing param for this source on this date
        update  param
        set     toDate = @dated,
                updTime = getdate()
        where   identifier = @identifier
                and param = @param
                and source = @source
                and fromDate = @vf


        -- add new value
        insert  param (  identifier, param, source,   fromDate, toDate, value,   updTime )
        select  @identifier, @param, @source, @dated, @vt, @value, getdate()


        -- if overwriting old data
        delete  param
        where   identifier = @identifier
                and param = @param
                and source = @source
                and fromDate = @dated
                and value != @value

    end

GO
```

4.2 Write Interface to Database (via stored procedure & Excel)

```
Option Explicit

'Necessary reference: Microsoft ActiveX Data Objects 2.8 Library [for ADODB.Connection]
'Necessary reference: Microsoft Forms 2.0 Object Library [for DataObject]
Dim Gcn As New ADODB.Connection
Dim GsServerName As String, GsDatabaseName As String

Sub sb_open_DB()
'Change History:
'Version Date       Programmer Change
'1.00    21/12/2010 Bernd      Create

If Gcn.State = 0 Then

   'Specify the OLE DB provider.
   Gcn.Provider = "sqloledb"

   GsServerName = "SBSERVER\SB_01"
   GsDatabaseName = "SULPROBIL"

   'Set SQLOLEDB connection properties.
   Gcn.Properties("Data Source").Value = GsServerName
   Gcn.Properties("Initial Catalog").Value = GsDatabaseName

   'Windows NT authentication.
   Gcn.Properties("Integrated Security").Value = "SSPI"

   'Open the database.
   Gcn.Open

End If

End Sub




Function sb_set_param(sIdentifier As String, sParam As String, sSource As String, _
                Optional ByVal sDated As String = "19000101", Optional sValue As String = "") As Boolean
'Stores data in database
'Change History:
'Version Date       Programmer Change
'1.00    26/08/2009 Bernd      Create
'1.01    03/06/2011 Bernd      Make 4th param ByVal because it get changed

Dim stSQL As String

If sValue = "" Then
   sValue = "null"
Else
   sValue = "'" & sValue & "'"
End If

If sDated = "19000101" Then
   sDated = "null"
Else
   sDated = "'" & sDated & "'"
End If

stSQL = "exec set_param '" & sIdentifier & _
```

```
        "', '" & sParam & _
        "', '" & sSource & _
        "', " & sDated & _
        ", " & sValue

'On Error GoTo errorexit

If Gcn.state = 0 Then
   Call sb_open_DB
End If

Gcn.Execute (stSQL)
sb_set_param = True
Exit Function

errorexit:
sb_set_param = False
End Function




Sub sb_delete(dtFrom As Date, dtTo As Date, _
            Optional sSource As String = "Markit")
'Delete database records younger than CdtFrom and older than CdtTo.
'Change History:
'Version Date       Programmer Change
'1.00    08/01/2011 Bernd      Create
'Const CdtFrom = #1/1/1900# 'Remember: #MM/DD/YYYY# is Excel's internal date format!
'Const CdtTo = #3/1/2011#  'Remember: #MM/DD/YYYY# is Excel's internal date format!
Dim stSQL As String

Debug.Print "From " & Format(dtFrom, "DD-MMM-YYYY") & " to " & Format(dtTo, "DD-MMM-YYYY")

stSQL = "delete from param where fromDate > '" & Format(dtFrom, "YYYYMMDD") & _
        "' and toDate < '" & Format(dtTo, "YYYYMMDD") & _
        "' and source = '" & sSource & "'"

Debug.Print stSQL

If Gcn.state = 0 Then
   Call sb_open_DB
End If

Gcn.Execute (stSQL)

Debug.Print "Finished."

End Sub
```

4.3 Read Interface from Database (via stored procedure & Excel)

```
Function sb_get_param(sIdentifier As String, sParam As String, _
            sDated, _
            Optional sSource As String = "Bloomberg") As Variant
'Retrieves data from database
'Change History:
'Version Date       Programmer Change
'1.00    21/12/2010 Bernd      Create

Dim stSQL As String
Dim vdbreturn As Variant

stSQL = "exec get_param ''' & sIdentifier & _
       ''', ''' & sParam & _
       ''', ''' & sSource & _
       ''', ''' & sDated & """

On Error GoTo errorexit

If Gcn.State = 0 Then
   Call sb_open_DB
End If

vdbreturn = Gcn.Execute(stSQL)
sb_get_param = vdbreturn(4)
Exit Function

errorexit:
On Error GoTo 0
sb_get_param = CVErr(xlErrValue)

End Function

Function sb_get_param_array(sIdentifier As String, sParam As String, _
            sDated, _
            Optional sSource As String = "Bloomberg") As Variant
'Retrieves data from database
'Return variant contains:
'1 - Identifier, for example "US912828HU78"
'2 - Parameter (field), for example "PRICE_MID"
'3 - Source, for example "Bloommberg"
'4 - Date, for example #12/23/2010#
'5 - Value, for example "100.32"
'Change History:
'Version Date       Programmer Change
'1.00    23/12/2010 Bernd      Create

Dim vdbreturn As Variant
Dim vreturn(1 To 5) As Variant
Dim stSQL As String

stSQL = "exec get_param ''' & sIdentifier & _
       ''', ''' & sParam & _
       ''', ''' & sSource & _
       ''', ''' & sDated & """

On Error GoTo errorexit

If Gcn.State = 0 Then
   Call sb_open_DB
End If
```

```
vdbreturn = Gcn.Execute(stSQL)
vreturn(1) = vdbreturn(0)
vreturn(2) = vdbreturn(1)
vreturn(3) = vdbreturn(2)
vreturn(4) = vdbreturn(3)
vreturn(5) = vdbreturn(4)
sb_get_param_array = vreturn
Exit Function

errorexit:
On Error GoTo 0
sb_get_param_array = CVErr(xlErrValue)

End Function
```

Note: The stored procedure would also be able to return an array of date / values pairs for a specified identifiers / parameters / sources if the date is left blank.